Chapter 4 Research Prototype

According to the research method described in Chapter 3, a schema and ontology-assisted heterogeneous information integration prototype system is implemented. This system shows that the integration method of this research is able to obtain the interoperability between multiple heterogeneous information sources. In this Chapter, the platform, architecture, and development of our prototype system is described in the following sections.

4.1. Prototype System Architecture

The prototype system architecture is shown in Figure 4-1. In our implementation, we tackle three kinds of heterogeneous information sources including structured data source, semi-structured data source, and unstructured data source. They respectively are relational database management system, native XML database, and web pages. Microsoft SQL Server 2000 is choose as the structured data source as well as Tamino 4.1.4 native XML Database is choose as the semi-structured data sources. Besides, we also take a web page repository as the unstructured data source.

Users connect to the prototype system which is built on the web server through the Internet by the client side browser. The prototype system receives the request from the user and sends it to the underlying information sources. After the underlying information sources process the requests, the prototype system collects the results and shows them to the user on the browser.

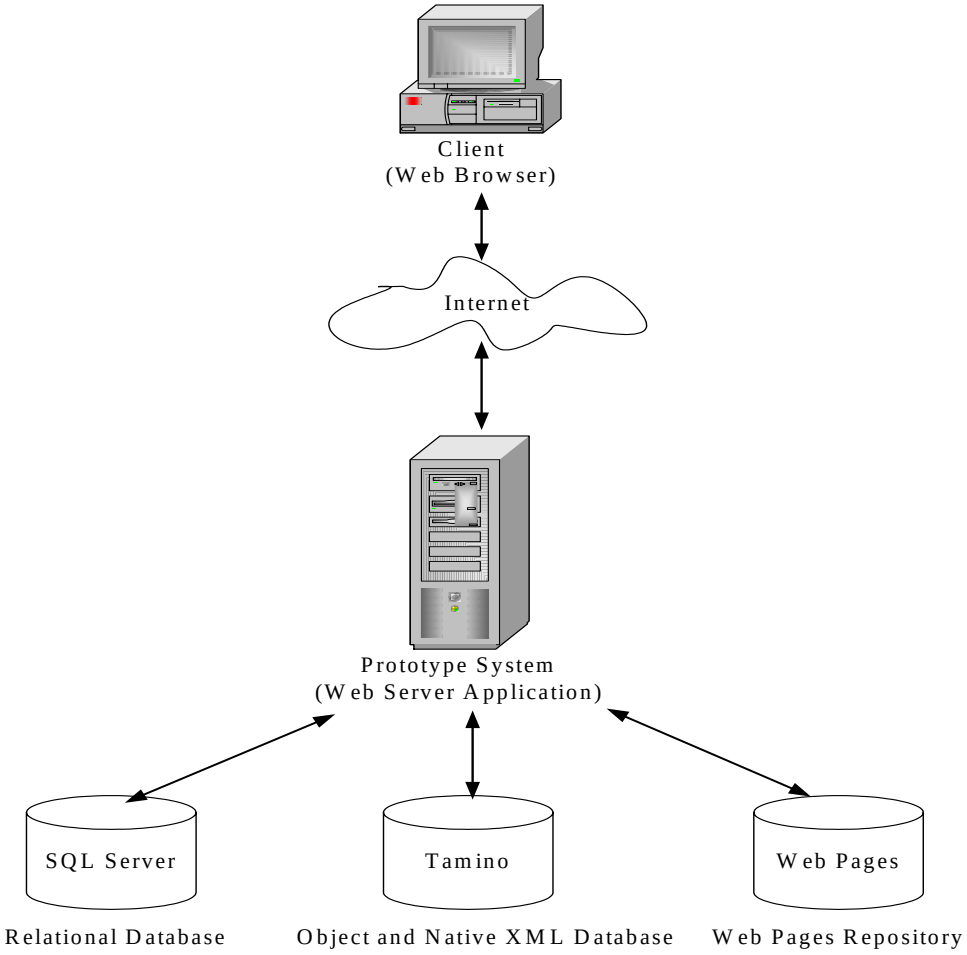

Figure 4-1: The Prototype System Architecture

4.2. Prototype System Platform

In our implementation, we choose Active Server Page (ASP) and Java Server Pages (JSP) as our programming language and Microsoft Windows XP Professional edition as the operating system. The specification of the prototype system platform is described in Table 4-1. We use Microsoft Internet Information Server 6.0 and Tomcat 5.0 as the web server. Microsoft SQL Server 2000 and Tamino 4.1.4 native XML database are the underlying databases. This prototype system uses client/server architecture. Microsoft Internet Explorer 6.0 is chosen as the client side web browser.

We also use Protégé 2.0 with OWL plug-in to edit the required ontology. And we use Jena API from HP Labs as the ontology inference engine to perform the needed reasoning in the prototype system.

4.3. Prototype System Design

Because of the lack of real world cases, we choose an application scenario of a university to implement our prototype system. Under such a scenario, we simulate three kinds of information sources as the required implementation scenario, which are:

1. Structured information source: Relational Database
2. Semi-structured information source: Native XML Database
3. Unstructured information source: Web Pages Repository

The complete schemas of the structured and semi-structured information sources are put in A Simple Example Implementation - A and A Simple Example Implementation - B. After the simulation of the required scenario, we start to create the needed global schema. According to the research method described in Chapter 3, we first use the generic construct correspondences to transform the schema of the structured information source, the relational database, into the form of XML Schema (see A Simple Example Implementation - C). Since the form of the schema of the semi-structured information source, the native XML database, is XML Schema, we need not to do the transformation on it. Afterward, we create global schema manually by using the method described in Chapter 3 (see A Simple Example Implementation - D).

We still must create the required ontology to catch the relationships that can not be hold in the global schema. We follow the research method to create the ontology in OWL by using the ontology editor – Protégé 2.0 with OWL plug-in from Stanford. Figure 4-2 demonstrates the creation of the ontology by means of Protégé 2.0. The complete ontology in OWL is shown in A Simple Example Implementation - E.

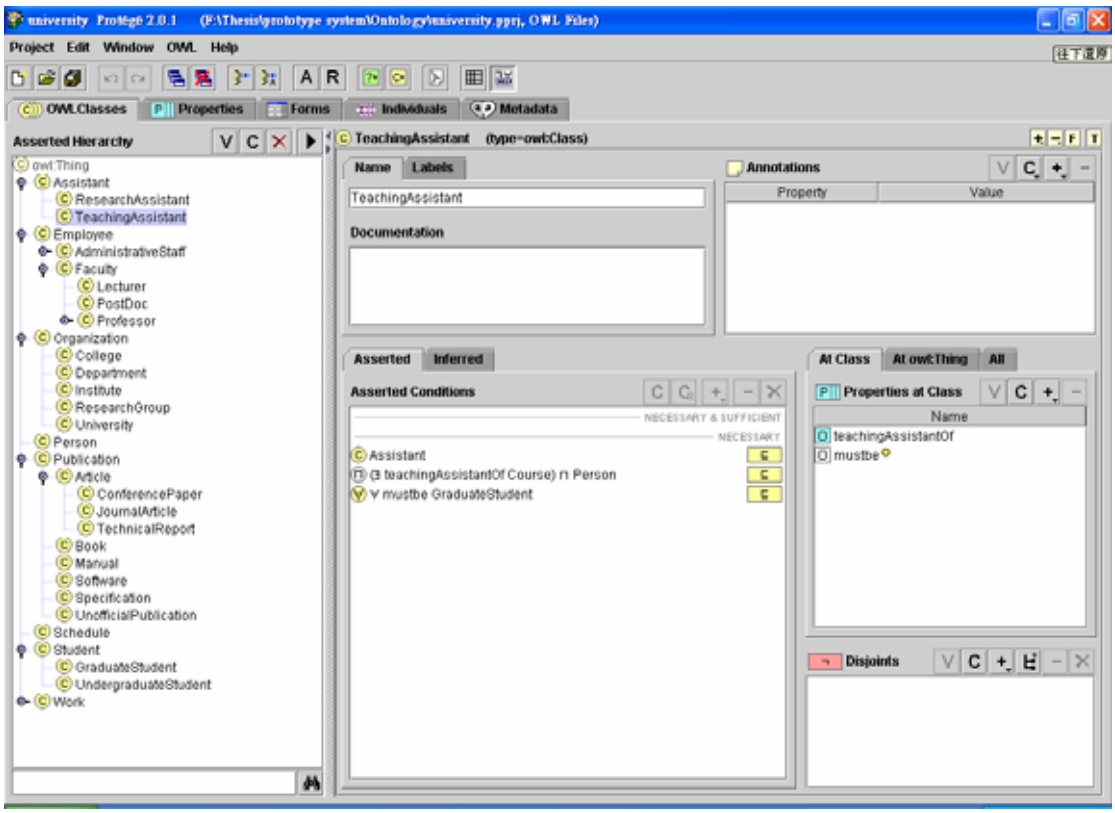

Figure 4-2: Demonstration of the creation of the ontology by means of Protégé 2.0

Figure 4-3 shows the prototype system functions design architecture. We provide an open query interface. Users formulate their query in the form of XQuery expression by referencing the structure described in the global schema. And the individual function of the other two primary query-processing components is described one by one as follows:

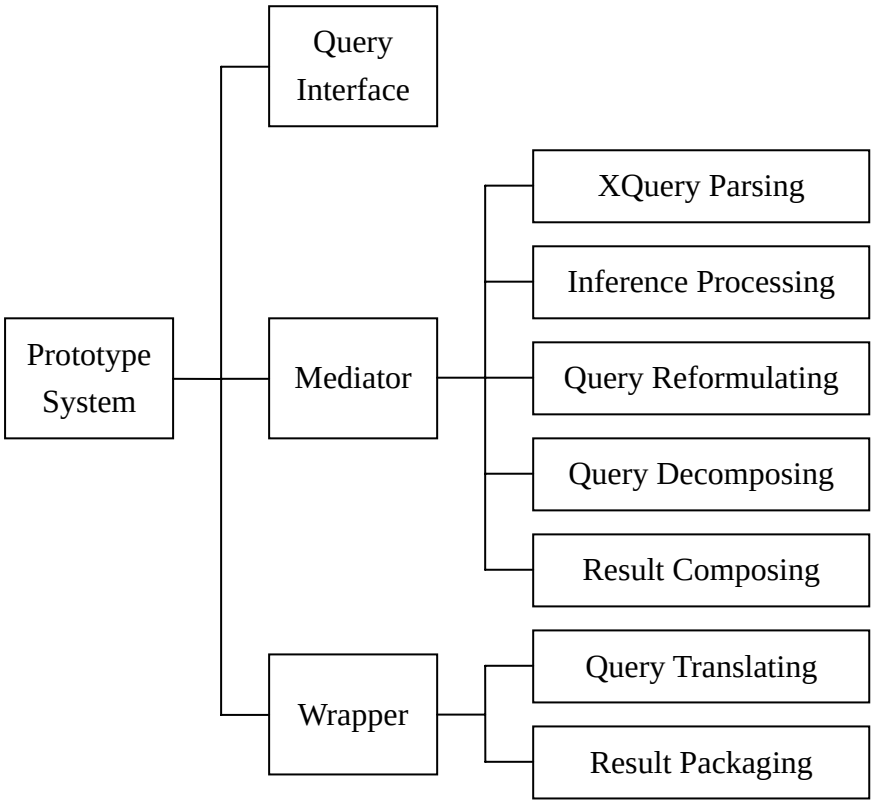

Figure 4-3: Prototype System Functions

1. Mediator:

I. XQuery Parsing:

Because the query interface is designed to accept open query in an XQuery expression, the system needs an XQuery parser to parse the query issued by the user from query interface.

II. Inference Processing:

After receiving the user query from interface, the mediator issues a reasoning request according to the user query to the ontology in order to find out the implicit relationships hidden in the user query.

III. Query Reformulating:

After receiving the reasoning results, the mediator reformulates the user query according to those results. Due to the reformulation process, the system can find out more answers with higher precision and accuracy to the query.

IV. Query Decomposing:

The mediator then arranges the query plan and decomposes the reformulated query into sub-queries according to the specified mapping which is specified according to the GAV approach.

V. Result Composing:

The mediator takes responsibility for composing the final results in XML document format from the temporal results sent by the wrapper. Finally, show the results to the user on the browser.

2. Wrapper:

I. Query Translating:

The wrapper takes the responsibility of translating the sub-query passed by the mediator into the form of the native query. And then pass the native query into local sources for finding out the needed answers.

II. Result Packaging:

The wrapper should also collect the query results sent by the local source and sent it back to the mediator to wait for further processing.

4.4. Prototype System Presentation

According to the functions description of the previous section, we demonstrate the implementation of the prototype system as follows:

Figure 4-4 is the query interface of this prototype system.

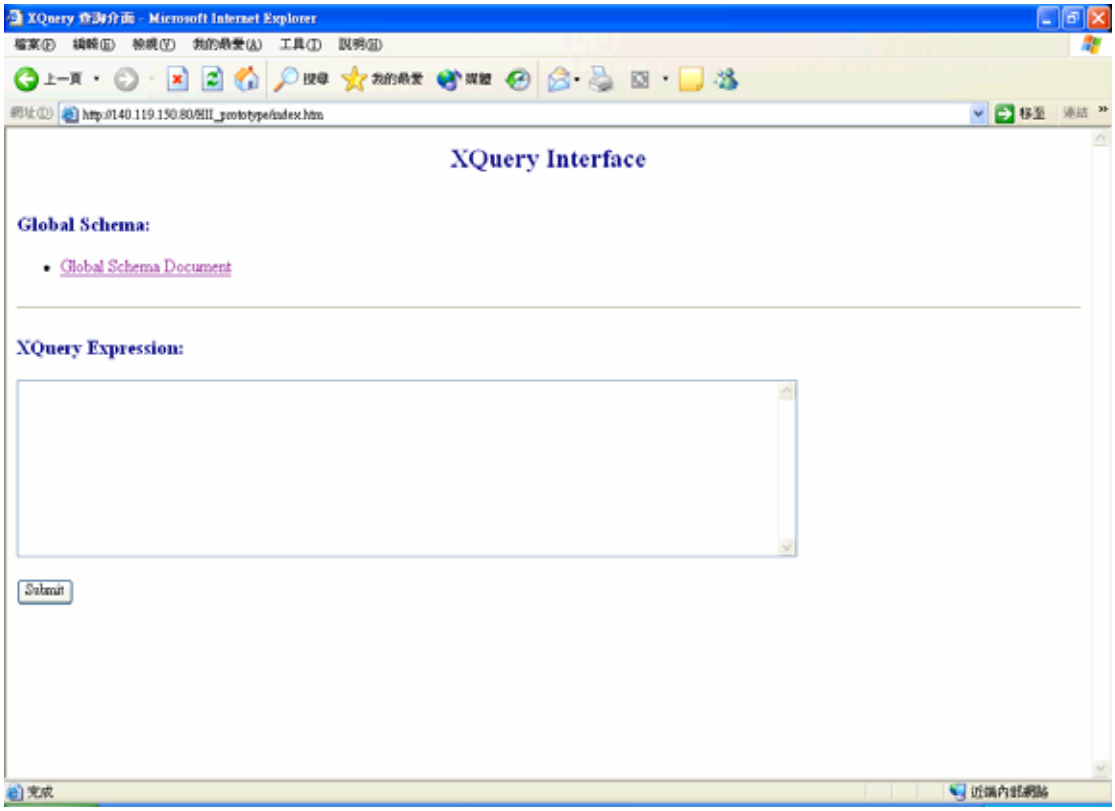

Figure 4-4: Query Interface of the Prototype System

Figure 4-5 illustrates that users formulate the query according to the global schema provided by the prototype system. We give a sample query here to illustrate the query process of the prototype system.

Sample Query: “find out advisors of teaching assistants of course ‘ADB’ ”

XQuery Expression:

```
FOR $a IN document('GS.xml')/GSROOT
LET $b := FOR $t IN document('GS.xml')/GSROOT/TeachingAssistant
          LET $d := FOR $c IN document('GS.xml')/GSROOT/course
                    WHERE $c/cname = 'ADB'
                    RETURN $c/ta_id
          WHERE $t/ta_id = $d
          RETURN $t/ta_name
WHERE $a//name = $b
RETURN $a//advisor, $b
```

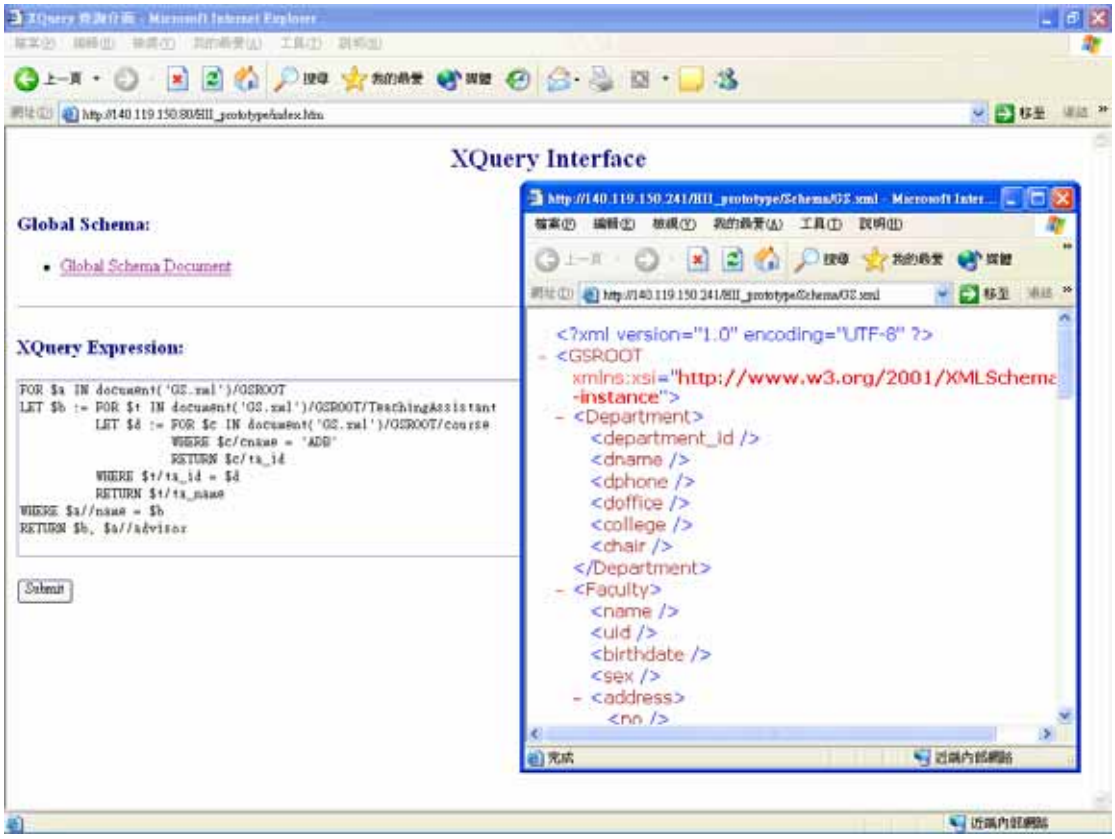

Figure 4-5: Users formulate the XQuery expression of their own queries according to the global schema

After receiving the user query, the system sends a reasoning request to ontology to find out if there have been additional relationships hidden in the user query. Then system reformulates the user query according to the reasoning result. Figure 4-6

shows the reformulated query generated by the system after the query reformulation process. The paths highlight in red color in the reformulated query is complemented by the system automatically according to the reasoning result.

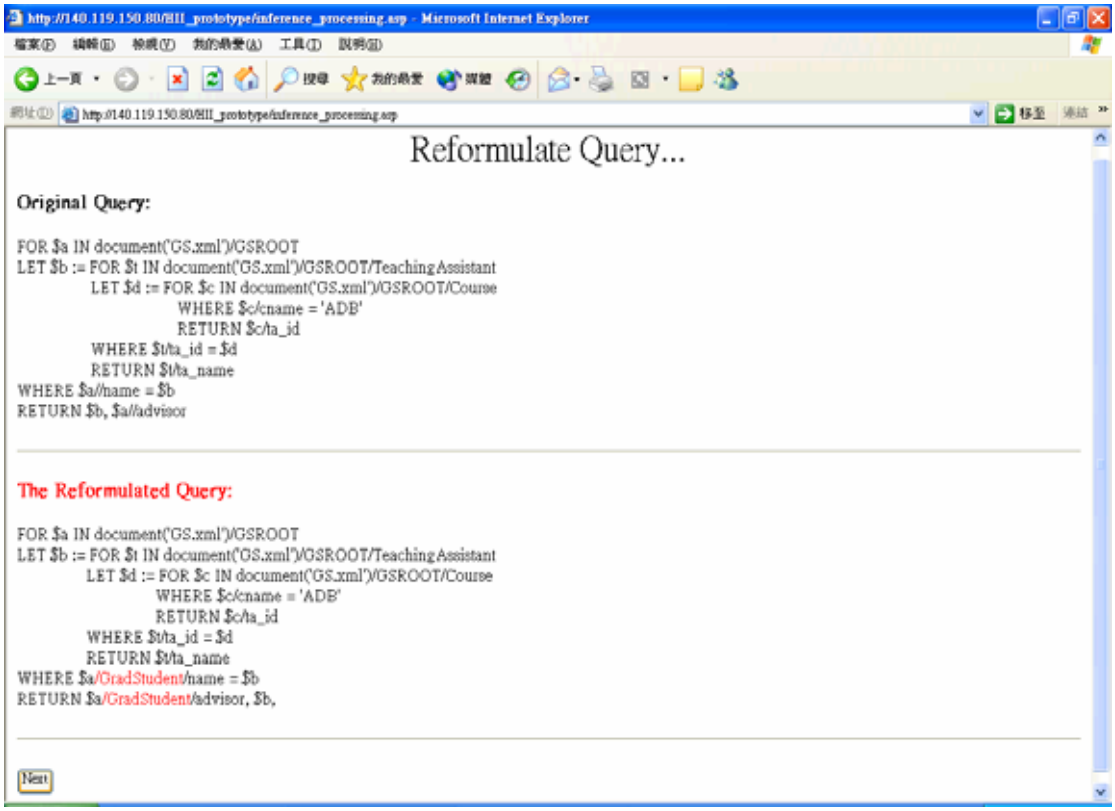

Figure 4-6: The reformulated query

The system decomposes the reformulated query and generates the query plan as the sequencial reference of sending the sub-queries to the wrappers. Figure 4-7 shows the query plan generated by the prototype system.

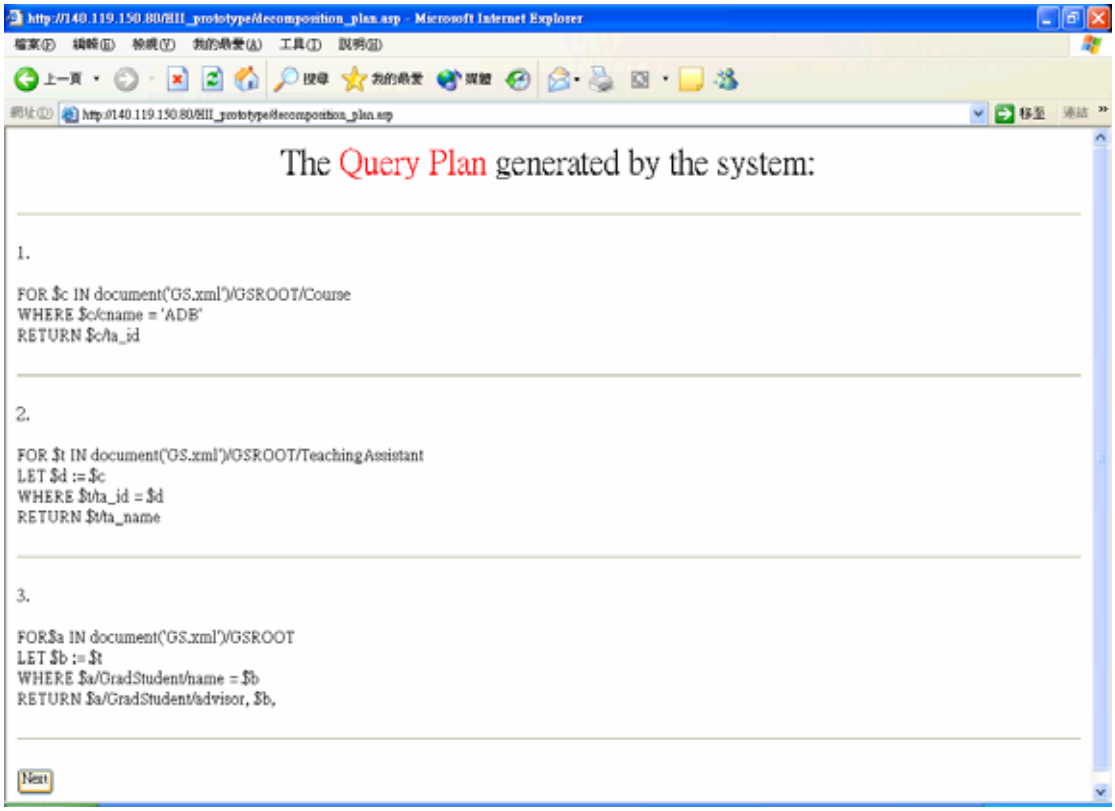

Figure 4-7: The query plan generated by the prototype system

According to the query plan, system passes the sub-queries to the corresponding wrapper. The wrapper takes responsible of translating the sub-query into the corresponding native query. Then the wrapper sends the native query to the underlying source to find out the answer of the query request. Figure 4-8 and 4-9 show the translated query generated by wrappers and the temporal result after querying the underlying source.

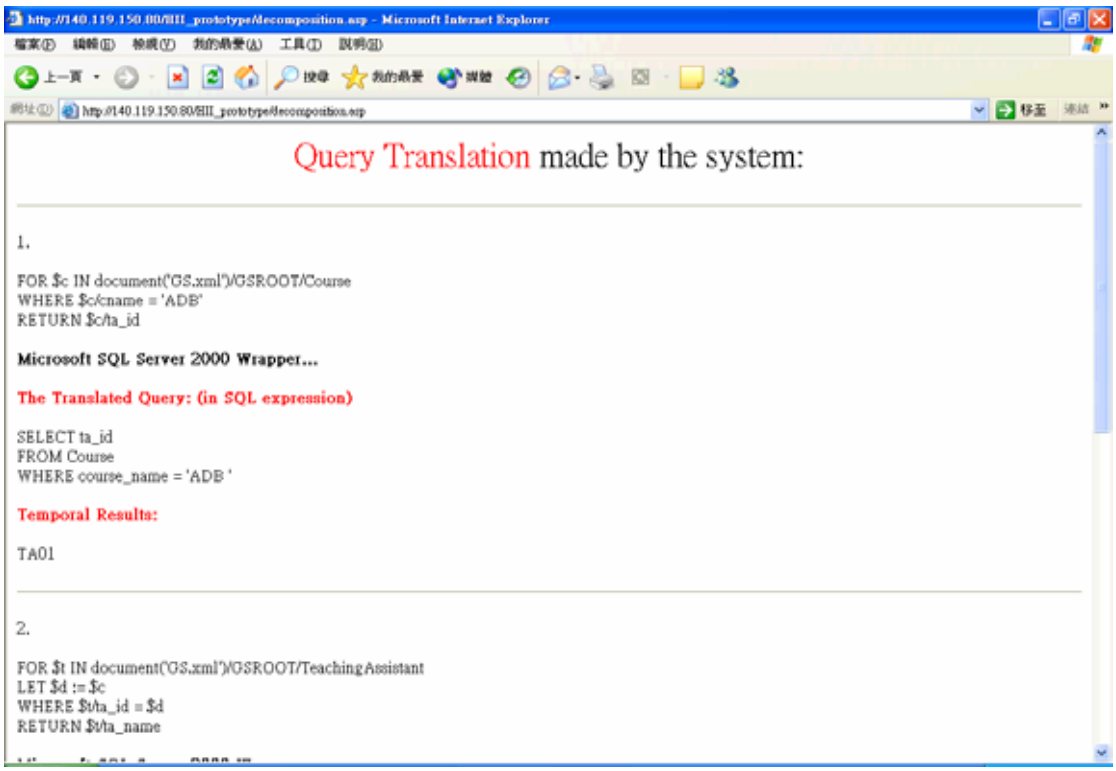

Figure 4-8: The decomposed sub-queries and the translated query generated by wrappers

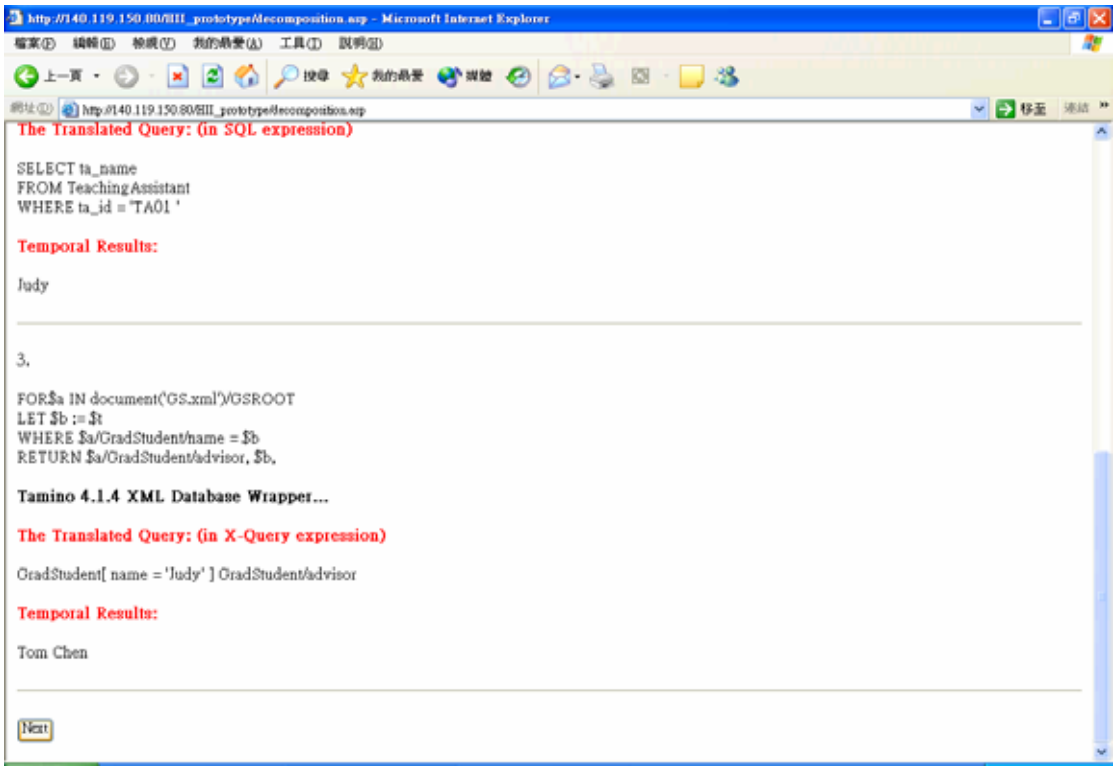

Figure 4-9: The decomposed sub-queries and the translated query generated by

wrappers (continue)

After finishing the tasks of the wrappers, the system collects the temporal results and composes them into an XML document to complete the query-processing. Figure 4-10 shows the completeness of the query-processing.

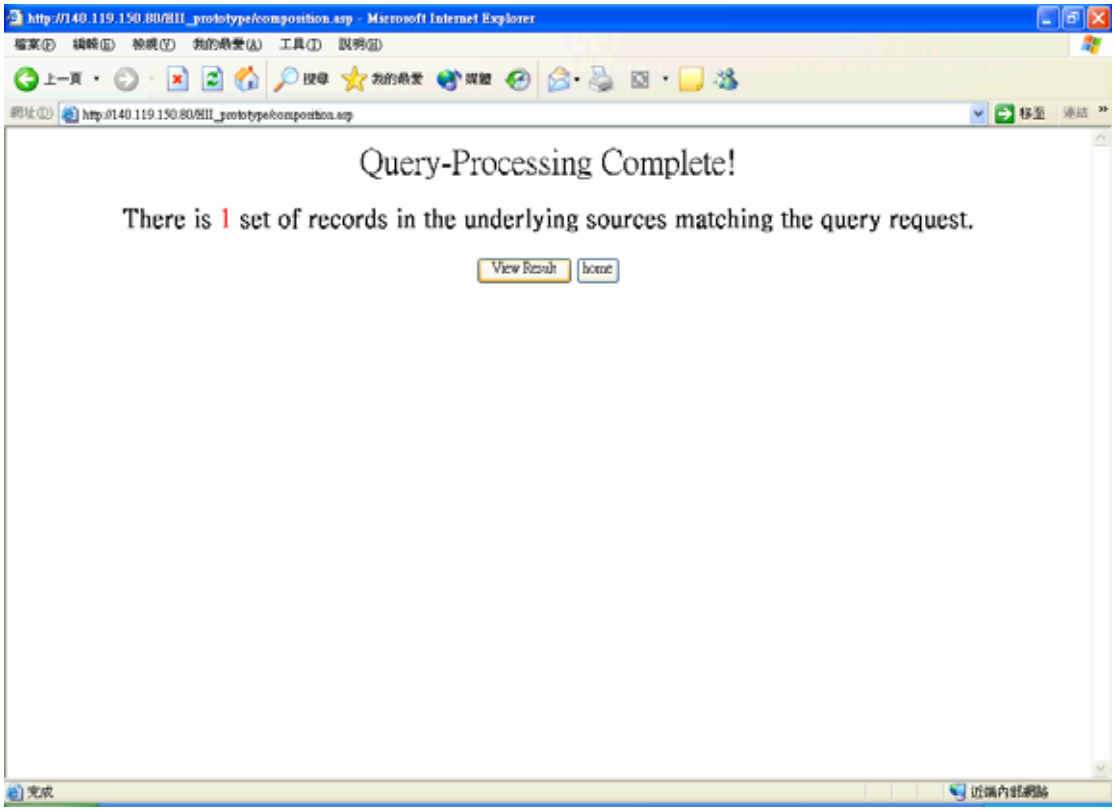

Figure 4-10: Query-processing complete

Figure 4-11 shows the final result in XML document. The structure of the final result is according to the global schema. We add an additional annotation attribute, source, for showing the answer that was retrieved from which information source.

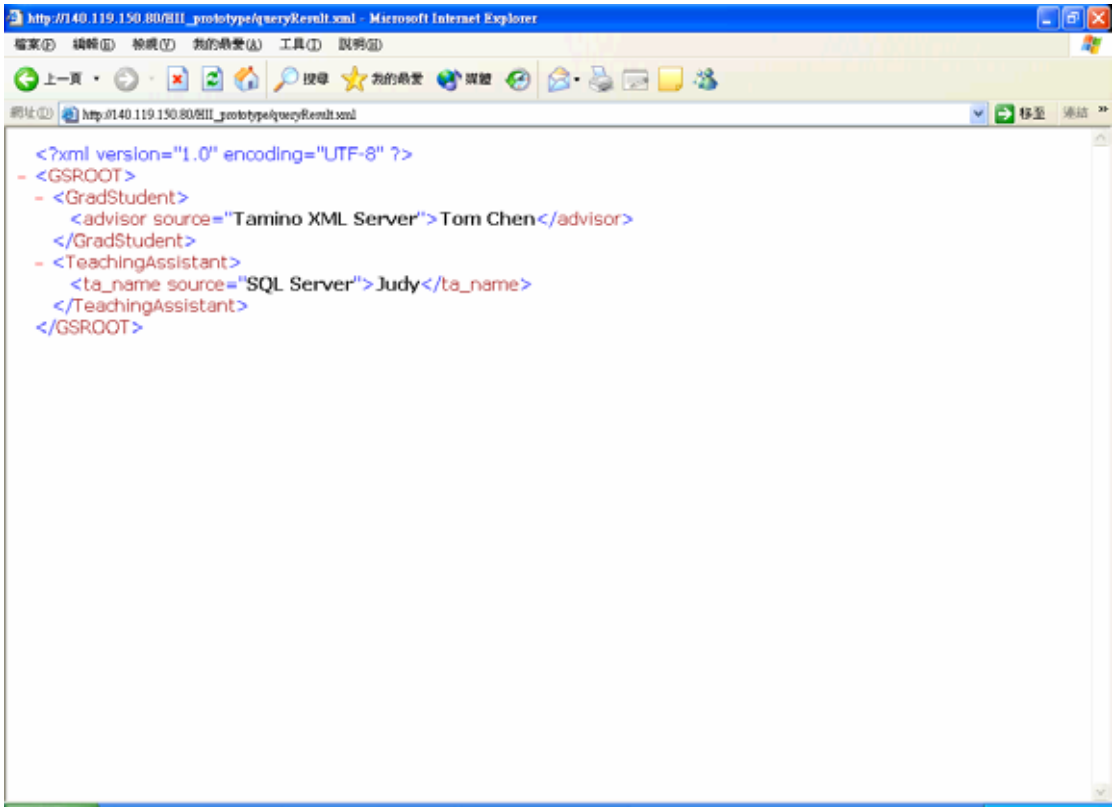

Figure 4-11: The query result in XML document